Version Control Systems and Subversion (SVN)

Paul Johnson

Nov. 24, 2010

1 Analogy For Microsoft Word Users

I often forget that the people I'm trying to help have a completely different experience than I do. I often plunge into a massive explanation that is completely useless. This section is my effort to avoid that.

Have you ever used "change tracking" in an MS Word document? That's handy because you can see what changes have been made, you can "reject" a suggested change and go back.

Version control is a bit like "saving your place" in a video game, actually. If your character gets killed, you can start again from a previous saved position.

Version control is somewhat like that, except it is more thorough. A version control system keeps track of all of your changes and allows you to "grab" any past version of a file. A version control system allows teamwork–several people can edit the same set of files at once and the system tries to reconcile the changes. The system tracks who makes changes. It asks them to explain the changes they make. And it allows rollbacks.

Version control is harder to set up the first time. It is not automatic.

2 Brief Background on Version Control

There are many different programs for version control. The time-honored standard was CVS (Concurrent Version System). Of all of the free software programs I have used, that one had 1) the longest run as the dominant, widely used program and 2) the best user manual. I suspect 1 was in large part driven by 2.

In the early 2000s, the software experts started to want more features, and a proliferation of version control systems emerged. Linus Torvalds, the author of the original Linux kernel, proposed a program "git". There are many others, "bazaar", "mercurial", and so forth. I have not used them.

At the current time, the version control system that is most like CVS is called SVN (short for Subversion). The commands that are used to interact with Subversion are almost the same as CVS, so as a simple user with simple needs, I don't notice much difference.

But, I have to admit, it is a hassle to get started with version control, but after you do it for a while, you will never want to go back. The alternative is to make a full copy of your project and set it aside frequently. If you have ever done that, then you know it has strengths and weaknesses.

You do need a "Subversion Server" running somewhere. I have Subversion installed in my personal computer, so I can just create a "repository" on my hard disk. If your system does not have Subversion, then you need to use a remote server. That sounds frightening, but it is not too hard. At the University of Kansas, we have Subversion installed on the cluster system known as

hpc.quant.ku.edu. Users can create their own personal SVN archives, or they can participate in the communal programming effort via SVN.

3 Conceptual Time Flow of Version Control

This document is not a substitute for a comprehensive SVN manual, but it should help us to get off the ground.

First: Create the "repository". That's the vault, where all the code and changes are recorded.

Second: Go somewhere else and download (or "check out") a working copy of the repository. Let's call that the initial working directory. That's like a sandbox. You can change whatever you want. Add files, add directories.

Third: Add those new files to the repository. That is called "committing" or "checking in" your files.

Fourth: Make sure the whole thing worked. Go to another computer, "check out" a new snapshot of your repository. Fiddle around with those files in the sandbox. When you are ready, you "commit" those changes into the repository. The repository keeps a "current" version and it also keeps the previous versions, which you can recover if you want to.

Fifth: Go back to the initial working directory and bring it up to date. You should see that the changes you made from your other computer will appear in this updated working directory. You can fiddle around with those files, commit the changes, and walk away.

The SVN repository, of course, should not be deleted. But you can feel free to delete any working directories. This means that, if you take some project and make a bunch of mistakes, you don't have to worry. You can always check out the repository again. If you want to, you can check out the repository as it was on some date in the past.

4 SVN: Why bother

> From Chad Perrin, "Use open source Subversion for personal document management," March 14, 2007 `http://articles.techrepublic.com.com/5100-10878_11-6167205.html`
>
> "A mechanism for automatic revision history management is probably most likely to be familiar to non-programmers because the most famous examples of wiki software employ such a technique for tracking changes to content and allowing undesirable changes to be reversed.
>
> As part of the revision history mechanism, a version control system such as Subversion not only maintains a central data repository copy of the current version of files that

have been entrusted to version control, but also maintains a log of changes that have been made from the present all the way back to the moment the files entered version control. Anyone who has been doing software development work for very long should be able to tell you how important the ability to roll back a file to a known-good state can be. This is in fact the central feature of any version control software: the primary reason it exists.

Subversion does this and much more. For instance, it also provides the ability to resolve version conflicts when two people have been editing the same file at the same time. In the real world, users who employ good practices such as making regular commits when working on files in version control, and updating local copies before committing changes, rarely run afoul of others' work. That rarity is nonetheless accounted for by Subversion, with conflict resolution features built in. It also supports easy branching of modified versions of the main development trunk, merging of divergent development branches, varying levels of checkout and update permissions for various classes of user, and a number of other useful features that project managers often find invaluable.

Personal document management

Another benefit of version control systems is that they allow you to work on a single project from a number of different locations, using a number of different computers, without having to keep any USB storage devices or CD-RW media on you at all times. As long as you have a version control client installed on the computer where you're going to work and have access to the server where the version control magic happens, you can check out the current version of the project and get to work.

Because of the fact that multiple copies of the same data are automatically synchronized to the same state when the checked out copy is updated on multiple client machines, a version control system like Subversion can also serve as an excellent backup system for a collection of files. This covers your everyday personal documents as well as source code; that is, if you interpret "project" to mean any relatively small collection of data–small enough so that you don't require a bandwidth optimized weekly backup to minimize the time spent copying your data. A personal documents directory usually fits this description perfectly, especially when you don't keep many files that tend toward multiple-megabyte file sizes (such as music, video, and high resolution image files).

If you are the type of computer user who understands that regular backups are extremely important as a precaution against hardware or file system failures, but just find yourself putting off regular backups because of the effort involved in configuring a traditional backup system or copying data to huge stacks of CD-R media, Subversion could be just what the doctor ordered. The simplicity of a tool like Subversion for personal document

backups can save you from yourself, or at least from your own tendency to procrastinate, and all you need is a second computer running the Subversion server software.

Because Subversion is not tied to a single, purpose-specific graphical user interface the way many proprietary systems like Visual SourceSafe and ClearCase are, it is easily adapted to nonstandard uses such as standard document control as well. You can still have your GUI environment, however, because there are a number of stand-alone GUI clients for Subversion, and Subversion has been integrated with a number of other GUI tools, such as Eclipse and even Microsoft's Windows Explorer file browser, via the TortoiseSVN client."

5 Create Your Repository "over there" on HPC.

Let's experiment in a safe way, where it is easy to erase mistakes and erase them and start over. We will create a repository within your personal user account.

SVN is already installed and configured on hpc.quant.ku.edu. If you happen to google and find a bunch of instructions about setting up "Apache" or Unix user groups or svn, just ignore that part. We did it already. You just need to use it.

From what I can tell, it will be necessary for you to actually log into hpc and manually create the repository in your personal account. That is a very easy process, as I will demonstrate in the next section.

After that, you can use any SVN "client" program to "get" the files when you want them and "send" them back when you are finished. (More on that later.)

First, I will walk through the process of logging on to hpc.quant.ku.edu and creating the archive. Later, I will learn how to use the free Windows program TortoiseSVN and see what it can do.

5.1 Start a test repository on HPC.

This is a nice way to "get your feet wet" in Linux. It is not a completely stupid task, it is useful, and it is not too difficult either.

On a workstation, use "some terminal program." (Putty in Windows is OK, xterm or gnome-terminal in Linux is OK).

On a Linux workstation, I'd just open the terminal and type this to "go" over to HPC:

```
> ssh username@hpc.quant.ku.edu
```

For username, I put "pauljohn". It is not necessary to include "username@", but I try to remember to do that in case I'm logged in with a different user name. If you only have one login name on all systems, it will be OK to just let the system assume you always have the same name. Run

```
> ssh hpc.quant.ku.edu
```

If you are in Windows, there is a free program called Putty, and if you double-click the icon for that, and make sure the "ssh" button is selected, then it will give you the right menu you need to log in.

After giving a password, you see this:

```
Last login: Thu Sep 30 15:50:44 2010 from 129.237.46.125
Access to electronic resources at the University of Kansas is
    restricted to employees, students, or individuals authorized by the
    University or its affiliates. Use of this system is subject to all
    policies and procedures set forth by the University located at www.
    policy.ku.edu. Unauthorized use is prohibited and may result in
    administrative or legal action. The University may monitor the use
    of this system for purposes related to security management, system
    operations, and intellectual property compliance.
[username@hpc ~] $
```

The dollar sign is the prompt. Type there!

One silly preliminary. Run this:

```
$ export SVN_EDITOR=nano
```

The SVN system will want to know what editor you will use when you need to interact with it. I suggest the simple editor "nano" for now. "nano" is based on the editor that was used in the email system pine, which virtually everybody at KU was using in the late 1980s and 1990s. If you don't do this, SVN will get mad and it won't work right.

If you don't already have a temporary directory "tmp", make one:

```
$ mkdir tmp
```

Then change to the tmp folder as the working directory.

```
$ cd tmp
```

Create an SVN repository for testing. Mine is named PJtestsvn

```
$ svnadmin create PJtestsvn
```

PJtestsvn appears as a directory inside my tmp folder, which is in my $HOME, so the full path to it is "/home/pauljohn/tmp/PJtestsvn".

The command "cd" by itself bumps you back to your home folder. See what I mean? Run the command "pwd" to see where you are.

```
$ cd
$ pwd
```

5.2 Import some content into the SVN repository

Now create some empty files somewhere so we have something to test with.

```
$ mkdir TmpWorkDir
$ cd TmpWorkDir
$ touch rawdata.txt
$ touch coolcode.R
$ touch something.txt
```

The "touch" command has the effect of creating an empty file if none exists, or, if one does exist, it gives it a current time stamp.

We tell the svn repository that we want to add these files in the repository's top folder.

```
$ svn import -m "initial" \
svn+ssh://hpc.quant.ku.edu/home/pauljohn/tmp/PJtestsvn
```

When that works, you see this output

```
Adding rawdata.txt
Adding coolcode.R
Adding something.txt
```

About my svn command. The backslash ("\") is only needed because my command ran onto a second line. It is not needed if yours fits on one line. The option -m "initial" is optional. If I don't do that, then the svn system wants to make me open an editor and type out an explanation of what these files are. The -m "initial" option just lets me give the message "initial" so I will be able to tell in the future this is the initial check in.

5.3 Check out a Sandbox Working Copy

Now test that the repository works. Back out of "TmpWorkDir." We might as well stay in character and create "TmpWorkDir2" to use for another sandbox.

```
$ cd ..

$ mkdir TmpWorkDir2

$ cd TmpWorkDir2

$ svn checkout svn+ssh://hpc.quant.ku.edu/home/pauljohn/tmp/PJtestsvn
A PJtestsvn/rawdata.txt
A PJtestsvn/coolcode.R
A PJtestsvn/something.txt
Checked out revision 1.
```

```
$ ls
```

you should see that a new directory called PJtestsvn was created. That's your "working copy" of the repository. Change into that directory

```
$ cd PJtestsvn
```

The program "nano" is a text editor. It is patterned after the editor that was used in an email program called "pine" that we used to use in the 1980s.

```
$ nano rawdata.txt
```

Put some stuff in there (anything you want), save it (Control O), close nano (Control X). I just put in gibberish like "iasdf ajsdkfl; ..." to see that the svn system works. Then commit it to the repository.

```
$ svn commit –m "Some random characters I chose"
```

The -m option gives a check in message. If I don't do that, svn will aske me to interactively type in change log. I don't mind that, but you might find it confusing at this point. The result is:

```
Sending rawdata.txt
Transmitting file data .
Committed revision 2.
```

Back out of there

```
$ cd ..
```

5.4 Lets Beat The Example Into the Ground Completely

Create yet one more working directory. We will test the newly uploaded file.

```
$ mkdir TmpWorkDir3
$ cd TmpWorkDir3
```

This downloads a current snapshot of the repository.

```
$ svn co svn+ssh://hpc.quant.ku.edu/home/pauljohn/tmp/PJtestsvn

A PJtestsvn/rawdata.txt
A PJtestsvn/coolcode.R
A PJtestsvn/something.txt
Checked out revision 2.
```

Use "cat" to display the contents of the file "rawdata.txt". See, it is still the same bunch of crap I started with:

```
$ cat PJtestsvn/rawdata.txt
iasdf ajsdkfl;ajdsf ;
```

```
asfdl;jaskf;aj
asdfk;asjdf;
```

Good. The file is there.

Then I edit rawdata.txt with nano

```
$ cd PJtestsvn
$ nano rawdata.txt
```

and put in some different crapola. Then I send it to the repository.

```
$ svn commit -m "some random crapola"
Sending rawdata.txt
Transmitting file data .
Committed revision 3.
```

Now, If I go to the other working directory, and run

```
$ svn update
```

It should find the new information in the repository and integrate it with your current working version.

5.5 Add Files and Directories to the Repository

The only way to add material in a repository is to check out a working copy, and then make the desired changes, and then add the new files to the repository.

Lets try to add a directory. I am currently still in the last sandbox, /home/pauljohn/tmp/TmpWorkDir3. (Run "pwd" to make sure). Check what I've got so far:

```
$ ls
PJtestsvn
```

Good, that's the copy of the repository. Change in there:

```
$ cd PJtestsvn/
```

It appears I have the right stuff:

```
$ ls
coolcode.R
rawdata.txt
something.txt
```

Create a new folder

```
$ mkdir SomethingElse
```

Add that to the repository

```
$ svn add SomethingElse
A SomethingElse
```

Go into SomethingElse and create a file

```
$ cd SomethingElse
$ touch anotherFile.txt
$ cd ..
```

```
$ svn commit –m "here's something" SomethingElse
Adding SomethingElse
Committed revision 4
```

```
$ cd SomethingElse
```

```
$ svn status
? anotherFile.txt
```

That means the svn system does not recognise "anotherFile.txt". I can manually add that particular file, though.

```
$ svn add anotherFile.txt
A anotherFile.txt
```

And then commit this version

```
$ svn commit –m "here is some file" anotherFile.txt
Adding anotherFile.txt
Transmitting file data .
Committed revision 5.
```

I think it is useful to see that we can add an individual file and check it in with commit, but usually I don't deal with files individually. Usually, I create a folder of material, and I want to add everything in the repository. That's useful when there are a lot of files involved. This should do it. The option "–depth infinity" on the add command means that svn will absorb "SomethingElse" and everything that is inside it, and everything that is inside everything inside that, and so forth.

```
$ cd ..
$ svn add ––depth infinity SomethingElse
$ svn commit –m here is everything that has been updated or added
```

6 Accessing Your Personal Repository from MS Windows with TortoiseSVN

A free program called TortoiseSVN is available. It is easy to install, and it runs as a Windows File Explorer "addon."

After installing TortoiseSVN, I wondered if I could recover the files from my repository.

I made a Windows folder, navigated into it, and then right clicked on the background. There should be two TortoiseSVN related options.

Click the one with the little arrow by it, and choose "Repo-Browser". We can use that to go see what we left on HPC.

"svn+ssh://pauljohn@hpc.quant.ku.edu/home/pauljohn/tmp/PJtestsvn"

The only *really annoying* part of this is that it asks me for my password *three times* in a row.

You can navigate the repository in the usual way, and when you get to the directory that you want to work with, right click and choose "check out". TortoiseSVN will then ask you where you want to keep your working copy. I chose "C:\Users\pauljohn\Desktop\whatever".

The current versions of the SVN files are downloaded and I could edit them.

When I created a folder, or files in a folder, I noticed I could right click, choose the TortoiseSVN commit, and then a menu appeared asking me if I wanted to add some files or directories to the SVN repository. I did so, it uploaded them.

Then I went to another system, grabbed the same repository, and the new stuff was in it.

Problem solved.

Because TortoiseSVN is point-and-click inside the Explorer, it is not so easy to tell you exactly what to do. But, I suppose the point of that is that you should be able to figure it out on your own if you can point-and-click at it.

7 Sharing a Repository with Other Users

This is the way "big time" software development works. People expect cooperative work effort on a common set of files. In the next section, I describe the steps I've taken to create a new repository in my user share on hpc to allow other people to download my course notes. I probably will never give other people permission to write in that folder, but I don't mind sharing.

If I want a folder that other people can both read and write, then I need to be a bit more careful. On HPC, we have set aside a folder in the common storage system for this purpose. The system-wide SVN repository is stored in /projects/svn. As of November, 2010, the following SVN registered projects exist:

```
hpcexample
md
```

HPC example is a collection of programs that use cluster computing.

md is the "missing data" simulation project.

Ordinary users in the HPC system are not allowed to create new project folders, but they can make requests for new projects to "clusterhelp@ittc.ku.edu".

The permissions on those folders are set as follows:

```
drwxr-xr-x 6 pauljohn pauljohn 8 Jun 2 11:53 hpcexample
drwxrwxr-x 6 pauljohn mdgroup 8 May 7 2010 md
```

The "hpcexample" folder is owned by pauljohn and the group is pauljohn, that means, as it currently stands, only pauljohn can write in there, but other system users are able to read that material. That means any user in HPC can check out "hpcexample," but cannot commit changes to it. Rather than repeat myself about access to "hpcexample", I would refer the reader to the web page where the details have already been committed.

`http://pj.freefaculty.org/cgi-bin/mw/index.php?title=Cluster:Main#A_Collection_of_Simple_Working_Examples_Using_Qsub_and_Multi-core.2Fthreading.`

Perhaps, at some point in the future, there will be other qualified users and a group can be created to make changes in hpcexample.

The "md" folder is owned by pauljohn, but its group is mdgroup. Note the group permissions are "rwx", so that anybody in the mdgroup can check in changes. The permissions for others are "r-x", which means that other people are allowed to check out a copy of the archive, but they cannot write changes back onto it. People who want to make contributions will have to request membership in the Unix user group called "md."

To grab a snapshot of the md directory from a Linux workstation, this command should do it:

```
svn co svn+ssh://pauljohn@hpc.quant.ku.edu/projects/svn/repos/md
```

If you are in Windows or a Mac system, your client program will want the options in a slightly different format, but the key idea is the same. The address you need to check out is probably going to be "`svn+ssh://pauljohn@hpc.quant.ku.edu/projects/svn/repos/md`".

8 I'm Shooting with Real Bullets Now

Recently I realized that my course writeups, my handouts, my slide shows, have degenerated into a completly unmanageable mess. There are different versions floating about, I can't remember which version of my regression handout is current, etc.

I decided to start an SVN repository on hpc where I would re-organize my writeups. I decided to put this archive in the /crmda share, where members of the crmda group can check out working copies if they want to.

Here's the embarrassing part. Even though I wrote this guide only two weeks ago, I had to read it step by step to remember how to do this.

So first I logged into hpc.quant.ku.edu, went into my folder, re-set permissions so group members will have read access

```
$ cd /crmda/users/
$ chown pauljohn.crmda pauljohn/
```

Unfortunately, that seems to give everybody in the crmda group write permission on my folder, so I need to tighten things up. First, I need to revoke write permission from the group. I'm doing this recursively, hence the capital -R option:

```
$ chmod –R g–w pauljohn/
```

I also want every file I create within there to have the group assigned as “crmda”, and the way to do that is to assign the “sticky bit” for the group marker on the folder “pauljohn”.

```
$ chmod g+s pauljohn
```

I ran “ls -la” to make sure I had this tightened down. In the end, what I want to see is the following:

```
$ cd pauljohn
$ ls -la
drwxr-s--- 10 pauljohn crmda 10 Nov 24 11:58 .
drwxr-xr-x 113 root admin 113 Nov 22 13:03 ..
drwxr-xr-x 3 pauljohn crmda 13 Apr 13 2010 acadiau
drwxr-xr-x 6 pauljohn crmda 7 Nov 1 19:19 RandBlas
drwxr-xr-x 6 pauljohn crmda 8 Nov 24 11:26 SVN-repo
```

The current working directory is now “/crmda/users/pauljohn”. That is where I will create the svn repository called “SVN-repo”.

```
$ export SVN_EDITOR=nano
$ svnadmin create SVN-repo
```

That creates a repository in a folder “SVN-repo” that is sitting in my share on hpc. I can access that from anywhere to “check out” a working copy, and then check back in my changes.

Now I go through the check out step on my laptop. That just first place where I want to use those files. I’ll also make checkouts on my home PC, my PC in Blake Hall, and probably on some workstations in the CRMDA.

On the laptop, I changed to my directory where I keep course material, and I run the command to check out the whole hierarchy under SVN-repo.

```
$ cd ps
$ svn co svn+ssh://hpc.quant.ku.edu/crmda/users/pauljohn/SVN-repo
pauljohn@hpc.quant.ku.edu's password:
pauljohn@hpc.quant.ku.edu's password:
Checked out revision 0.
```

I’ll test out the process of adding new content before I quit. In that working copy on my laptop, I want to start a folder for documentation projects I’m working on. The first document I put into the new folder will be *this file* you are reading right now!

```
$ cd SVN-repo
$ mkdir HOWTO-docs
$ cd HOWTO-docs
$ cp ~/SVN-intro.lyx .
$ cd ..
```

The "cd .." command takes me back to the main repository directory, where I see "HOWTO-docs" as a sub directory. I want to add that, and all of its files, to the repository.

In the previous section, I noticed that when I added a directory, svn did not automatically add the files inside the directory into the repository. So I individually added the files. Since then, I've re-read the manual and I've learned that the add command can work recursively if I add the "depth" option.

```
$ svn add --depth infinity HOWTO-docs
A HOWTO-docs
A HOWTO-docs/SVN-intro.lyx
```

That adds the directory and files to the repository framework, but it does not upload them. We have to force a commit or "check in" like so (ci = commit).

```
$ svn ci -m "initial"
pauljohn@hpc.quant.ku.edu's password:
Adding HOWTO-docs
Adding HOWTO-docs/SVN-intro.lyx
Transmitting file data .
Committed revision 1.
```

Various programs I use, like Emacs and LyX, are SVN-aware, which means they have menus to check out files and check them back in to the repository. I often prefer to avoid those conveniences; to feel sure things are working, I'd rather just type svn commands in the terminal. But your mileage may vary.

Now, if you want to grab your own working copy of my SVN repository, go to some computer and tell it you want it. If you are a member of crmda on hpc, you can have it. Run this on a Linux workstation:

```
$ svn co svn+ssh://hpc.quant.ku.edu/crmda/users/pauljohn/SVN-repo
```

If you are in Windows or a Mac, your interface will be different, but I'm pretty sure the part the program will need is `svn+ssh://hpc.quant.ku.edu/crmda/users/pauljohn/SVN-repo`.

Now I could, with a completely clear conscience, throw this laptop computer off a bridge and forget about it. Because I am secure in the knowledge that the file "SVN-intro.lyx" is saved in my repository on HPC. All I need to do is get another laptop, check out a new working directory, and I'm back in business.

9 There's a Lot More to Do & Learn

Subversion is an industrial-sized, team ready production tool. It can do all kinds of stuff most ordinary humans like you and me never really need. When a program is finished and ready for packaging, it can be tagged with a version number, and then exported. Revisions can be created on "branches," and changes can be merged back onto the main "trunk" of the project.

We do that kind of stuff in software development, but you probably won't need to if you are just using Subversion to keep track of your personal software and document development. If you decide to convert your R code into an R package, well, the additional power of SVN will help you out.

Finally, yes, I realize there are abbreviations and shortcuts that will work to make some of the commands here shorter. Inside one system, it is not generally necessary to access files through "svn+ssh" because a more simple access protocol is available, for example. But, if svn+ssh does work, why bother learning another style? For abbreviations, it is true that one replace "commit" with "ci" and "checkout" with "co". Run "svn help" to see a list of abbreviations.